SUMMARY

The theory of files is a tool for the logico-mathematical treatment of automatic non-numerical data processing problems, such as machine accounting, information retrieval and mechanical translation of languages. The main result which has been obtained so far from the application of the theory of files is the formulation of a simple pattern to which the data flow of any information processing procedure conforms, regardless of how many files are involved. The flow of each file can be controlled and coordinated with the flow of the other files by means of five boolean parameters, called 'indicators'.

A specially designed Algebraic Business Language exploits this result for the purpose of programming digital data processing systems. This paper also probes into the impact of the theory of files upon the logical design of digital information processing systems.

INTRODUCTION

The first step of any initiative yielding to the scientific knowledge of a new field consists of the definition and development of a language and of a notation able to describe the phenomena which characterize such a field. In particular, the adaptation and the adoption of one specific language - the mathematical language - has been successful in several areas where the need for a scientific investigation existed.

That aspect of human activity which seems to be growing the fastest (in such a way that it sometimes threatens to minimize the importance of all the others) is the control of paperwork. Paperwork is one of the most impressive products of civilized society; its relevance with respect to the other activities swells with the progress of our economy and technology in a way which is liable to jeopardize this progress itself. Paperwork is generally carried out by machines; however, the work of organizing, coordinating, defining and describing it is still performed by humans. The proportion of the total available manpower that it absorbs grows dangerously with the wealth and sophistication of our society. The phenomenon of paperwork control has reached the stage where it should be investigated scientifically, hopefully to repeat the success that comparable scientific approaches yielded when they were applied to other fields, in terms of promoting the knowledge, suggesting the development of suitable techniques, allowing the adoption of reliable procedures and collapsing the amount of work necessary for carrying them out. We believe that this can be done, and that the appropriate language for analysing coordinated paperwork can only be mathematics. The purpose of the theory of files is to support this belief.

Today the theory of files, whose basic definitions are stated in\(^1\), has been applied only to investigation of a specific area of paperwork control: the area of systems analysis, namely of the analysis and definition of those procedures which can be carried out by automatic data processing systems. The specific problem of non-numerical data processing has been emphasized, the main reason for this being the widely spread prejudice that this field cannot possibly be approached scientifically.

AVAILABLE COMPUTER LANGUAGES

A wide selection of computer languages designed with the aim of providing tools for automatizing the programmer's work is available today; however, it seems that none of these languages can help the systems analyst. We think that the main shortcoming of such languages (which range from simple assemblers to autocoders able to handle macros, and to such languages as the IBM Commercial Translator, COBOL, FACT, Flow-Matic or AIMACO), are all designed according to the pattern that we called 'v. Neumann Language'\(^2\): a v. Neumann language is a language in which a phenomenon is described by means of a sequence of statements divided in two categories - 'executable' statements and 'descriptive' statements - in such a way that the statements of the first category can be put into a one-to-one correspondence with a flowchart\(^3\) of the procedure involved by the phenomenon represented.

Such a language can be used successfully for describing layouts of information supports and sequences of actions, namely procedures. Unfortunately such language cannot possibly provide for a synthetical and compact definition of the compound of logical conditions to which any action is subjected, nor for the synthesis of a coordinated flow of information. For instance, if we consider a language such as COBOL and we try to use it for representing integrated data processing procedures, the following shortcomings come to light:

* From the collection of the Computer History Museum (www.computerhistory.org)
Each statement has the form "IF condition
THEN action", where the action denotes a
sequence of steps and the condition denotes
a boolean expression. Nevertheless the
execution of the action is only apparently
fully controlled by the condition, in the
sense that the value 'true' of the con-
dition is necessary but not sufficient for
the execution of the action. Such execution
depends also upon the path of the control
trough the procedure description, i.e., on
the result of several tests, some of which
may have preceded by far the action in
question. Since the possible path of the
control are myriad, then, for determining the
circumstances under which a certain action is
to be executed, one should carefully
trace through all of the procedure descrip-
tion. It is usually not practically feasi-
able even to identify such circumstances,
nor to correlate an action to the original
input information. What is needed is a
language by which all of the conditions
affecting an action are compounded into a
unique statement; such a language could not
dispose contain any control statements, and
consequently could not possibly be a v.
Neumann language.

Most documents are self-explaining, as far
as their path between different procedures is
concerned. However, one of the big
problems in systems analysis is the de-
termination of when and under which circum-
stances a document is entered into or issued
from a procedure. This well known problem of
efficiently coordinating the data flow becom-
es one of the main issues of systems analysis whenever one wants to save com-
puter time by increasing the degree of pro-
cedural parallelism. In COBOL, like in any
other v. Neumann language, the flow of in-
formation can only be controlled by means
of input-output statements, and by a proper
organization of the control statements, i.e,
by carefully planning the flow of the ma-
chine control through the procedure de-
scription. In such easy applications as
payroll, where the degree of parallelism
cannot possibly be high, COBOL can be used
successfully. On the contrary, consider
applications in which, for the sake of
saving computer time, several functionally
independent procedures that relate only by
the fact that they operate on sets of files
which are sorted with respect to the same
key (i.e., they are equiordered), are run
in parallel; then the use of such a lan-
guage leads to long and exceedingly in-
volved descriptions. Even in some compa-
ratively simple applications, where for
example, billing and accounts receivable (or
ordering and accounts payable) are run in
parallel, together with the updating of a
master file and with the preparation of data to be used later by the management
(such as notes for exceptional cases, re-
quested reports, or totals), COBOL can com-
pare favorably to some of the available
autocoders as a programming language. How-
ever, it does not seem to be an appropriate
analysis language for such applications.
In cases where the degree of parallelism is
high and the data flow is complex, such a
language should be discarded.

Last and least, it appears that the use of
some type of kindergarten English, whose
adoption seems to be due to the objection-
able assumption that it is more readily
understood by top executives than any more
appropriate technical notation, is an obsta-
cle to the use of COBOL even as a program-
ning language, because it yields comparati-
vably long procedure descriptions. However,
this last shortcoming is really irrelevant,
primarily because it affects COBOL only as
a programming language. Further this short-
coming can be removed easily from the lan-
guage without any major change in the logic
of its translators. It has also been a
common experience of individuals program-
ing with COBOL that after a few statements
one drops the English of the language and
uses abbreviations, especially for such
phrases as "IS GREATER THAN".

Basic Ideas

In addition to the trend which finally led to
COBOL, two independent ideas were developed in the past few years. Both aimed at the cre-
tation of a system language suitable for describ-
non-arithmetic data processing procedures.
The first philosophy can be summarized as fol-
loW: 'We must algebraize the non-numerical pro-
cedures, in order to be able to apply to them
successful algebraic languages such as Fortran'.
The other can be expressed in this way: 'The
major problem in non-arithmetic data processing is the one of defining and coordinating the
data flow: before we can design a system lan-
guage, we should discover and formulate the
laws of the data flow'.

The theory of files is but a synthesis of
these two ideas; from a methodological stand-
point the theory of files consists of expressing and analyzing algebraically the laws of the data
flow.

The starting point in the theory of files
was the remark that if we consider the merger
between two files of records as a sum, while
the merger of them with selection of all those re-
 cords whose key is not present at least once in
both files as a product, then, under certain
circumstances, sets of equiordered files can be
reduced to boolean algebra of files. In such
an algebra the most common file handling op-
erations can be defined by simple algorithms: for
example, a k-way sorting-by-merging procedure
(when fixed or variable length sequence) is
represented as a recurrent summation of k files.
From a file-theoretical standpoint, a procedure is broken down into a sequence of pulses, at whose beginning new records are (logically) entered, during which calculations are performed, and at whose end all the completely processed records are (logically) filed. Only records whose keys all have the same value are considered in a pulse. A sequence of pulses during which all the records of all the files involved in a procedure, whose keys equal a certain constant, are processed, is called a phase of the procedure.

The language that we propose for describing procedures is the Algebraic Business Language (ABL). It is described in1, where the basic concepts of the theory of files are defined mathematically. In its simplest version, an ABL procedure description consists of a sequence of "conditional expressions", namely of sets of executive orders ("actions") subject to boolean expressions ("conditions"). There are no control statements, and the conditional expressions are to be considered sequentially, i.e., from the first to the last.

Optimization.

Simplification. In each procedure, a logical order of the files involved must be given by the analyst.

Definition 1: "Let us denote by DD the data description of a given problem; then two procedures are DD-equivalent if they both transform any input organized according to DD into the same output".

Definition 2: "A procedure P is called DD-optimized if, in the space \{DD, P\} of all the procedures which are DD-equivalent to P, P both
a) Maximize the parallelism of logical input-output
b) Minimize the amount of internal processing".

From an applicative standpoint, only DD-optimized procedures should be considered; notice that the maximization of the parallelism of the physical input-output flow can be obtained only on the basis of a logical one whose parallelism is maximized.

Now two pulses are always independent as far as internal processing is concerned, and two phases are always independent as far as input-output is concerned; consequently, in order to DD-optimize a procedure, it is sufficient to
1) Maximize the parallelism of the logical input-output within the pulse.
2) Minimize the amount of internal processing within the phase.

Since ABL is a sequential language, point II can be accomplished simply by performing a precedence analysis and a simplification of the procedure description. (Notice that this would not be easy in a v. Neumann language).

In order to discuss point I, let us consider separately input-output.

Input. The pattern which maximizes the input-parallelism is unique for any phenomenon of the kind we are considering. More precisely, it consists of the following:
1) No more than one record per each file is entered during any pulse.  
2) Consistently with 1), a record belonging to any file is entered as soon as it is both logically available and all the records pertaining to the current phase, belonging to any one of the files which precede it in the logical order, have been entered.
3) A phase is over at the end of its pulse during which the last record pertaining to it is entered.
4) Logical input is only performed at the beginning of the pulses.

Since the input pattern is unique for any procedure of the type we consider, the analyst is not burdened with the control of the input; he can just forget about it. The only way in which the analyst using ABL can control input is by designating the logical order of the files properly.

Output. Unlike input, which is fully standardized and automatic, output is entirely and directly controlled by the analyst. In fact, the conditions under which documents are to be issued always depend upon the particular phenomenon considered. Furthermore, the determination of these conditions can often be considered as the major single factor in the representation of this phenomenon. Since it is important to have these conditions compounded in a single synthetic expression for each output file, the output of each file is controlled by a flow control expression, consisting of the name of the file in question followed by a boolean expression denoting the condition under which a record of this file is to be issued.

Indicators. The boolean variables used for writing a procedure description (which are compounded into boolean expressions in ABL, while they consist of sets of parts of different statements in any v. Neumann language) may have three origins:
   a) They may be generated by comparison between numeric, alphameric or boolean entities.
b) They may be determinations of conditional variables.

c) They may be references to the current configuration of the data flow.

While no need arises for a special discussion of the conditions of the first two categories, we may point out that when any v. Neumann language representation of a procedure is used, such references are generated by means of careful constructions and comparisons of keys. The theory of files suggests that the layouts of the keys of the files are given as part of the data descriptions, and that the keys of the records are constructed and related automatically to each other as part of the I-O operations; consequently, these operations are not under the control of the analyst. Since references to the current status of the data flow are often necessary for making decisions which condition the phenomenon considered, ABL must have a provision for giving to the analyst complete information about it. The configuration of the data flow never changes during any pulse, and from a file-theoretical standpoint it can be fully characterized by stating the occurrence or omission of five conditions for each one of the files involved. This can be done by means of INDICATORS, five boolean variables per each file, whose value never varies during any pulse.

Let us explain intuitively what each indicator of a file F stands for:

1) The 'EXISTENCE INDICATOR' of F denotes the logical presence of a record of F.

2) The 'LEFT DERIVATIVE' and the 'RIGHT DERIVATIVE' of F characterize those records of F whose key has a value which is different from the value of the keys of all the preceding (following, respectively) records of F.

3) The 'INPUT - OUTPUT INDICATOR' of F by being "on" denotes those pulses where a record of F is entered or issued.

4) The 'NON CONFORMITY INDICATOR' of F characterizes those records of F which are incomplete or non-conforming.

Four further indicators, which are common to all files, are available in each representation of a phenomenon for denoting its initiation and closure. No other information regarding the data flow is needed in any DD-optimized procedure, in whose description the indicators can be used without any distinction from the other boolean variables.

The setting and resetting of the indicators (i.e., the 'indicator logic') is performed automatically according to the rules stated in \( \text{section 3} \), where the laws of the automatic data flow control-in particular of the input mechanism - are stated in terms of relations between the indicators. The set of values of the indicators in each pulse is a synthesis of the data flow control related to it, and is obtained as a subproduct of the logical operations involved by the input and output.

Though the analyst can neither set nor resets any indicator, an indicator can be used anywhere in the procedure description: in particular in the Flow Control Expressions.

**Hardware and Implementation**

**Sequential Languages**

Like a mathematical synthesis of a physical phenomenon can be stated by means of a sequence of equations, so the theory of files allows one to express a mathematical synthesis of a data processing phenomenon in ABL by means of a sequence of conditional expressions. In both cases the sequence is considered from the first equation (or conditional expression, respectively) to the last one. The flow control expressions are conditional expressions where the action consists of issuing a record. Neither the equations of an algorithm nor the conditional expressions of a non-arithmetic procedure description are in a one-to-one correspondence with the steps of any path that a machine control would follow in order to carry them out. Unlike any v. Neumann language, ABL is 'sequential' and asynchronous with respect to the way the procedures described are implemented. Our study shows that languages having this structure are generally more suitable than v. Neumann languages for approaching data processing phenomena scientifically.

If one wants to utilize the theory of files not just as a method of investigation but also as a tool for the automation of systems analysis, he must be able to develop mechanically sequential outlines into flow charts i.e., into procedures. More precisely, one should be able to transform the sequential representation of any data processing phenomenon into a DD-optimized flow chart. Apparently this transformation can quite easily be made because of the standard input scheme, and of the fact that each conditional expression completely determines one specific issue of the procedure, like the presence of a record in an output file or the value of a certain field of an output record, etc. A difficulty arises when we consider the interrelationship between the indicators of the various files; for example, the condition-part of a certain conditional expression, say EA, may depend upon the setting of an indicator of an output file whose records are filed under the control of another Flow Control Expression, say EB, which comes after EA in the sequential description. Consequently, the sequence of operations must be properly arranged in order to avoid unnecessary look-aheads. Such rearrangements should not be performed by the analyst, who should only be concerned with the statement of the information processing effect of the phenomenon, rather than with procedural considerations or with any simplification of
the correlation among expressions. This simplification should be carried out by the machine, together with the entry and removal of auxiliary conditional expressions and with the optimization of the arithmetic formulae. This last operation should not be bounded to the optimization of each single formula within itself, but should consist of analyzing the relations between different expressions in order to avoid unneeded repetitions. The study of Semapraxis codifies the efforts of analysts toward the intelligent utilization of computers for such machine simplifications. In particular by Feldstein enunciates such details.

The ABL representation of a phenomenon can also be mechanically checked against tautologies and contradictions which may depend on an erroneous analysis of the phenomenon itself; for instance, in the above example, if the phenomenon is coherently stated, ES should depend neither directly nor indirectly on EA.

Special Devices.

Most stored program data processors are provided with an operating system which includes efficient buffered input-output subroutines. Some data processors-the IBM 7070-74, for example-have specific features (scatter-read-gather-write, highly parallel memory bus, block transmission with rearrangement, etc.) which allow the programming of very efficient I-O routines, including the necessary key logic. In accordance with the adoption of some new ideas in the design of machinery, (consider for instance the non-arithmetic processor of the IBM Harvest, or the systems with a Fixed+Variable structure) it is sometimes convenient to wire such routines, which become parts or modules of the hardware.

When a system has to carry out procedures represented in ABL, a similar alternative arises for the indicator logic, which will be programmed for standard systems and built for more advanced and specialized ones.

A third case where the issue of a comparison between wired and programmed 'giant commands' varies with the modernity and specialization of design of the basic hardware is related to the handling of the compact and flexible 'table operations' with whose use ABL provides the analyst (see section 2).

The implementation of ABL is significantly conditioned by the hardware; it appears more difficult to carry it out for standard, strictly stored-program computers than for more advanced ones. The generation of a program on the basis of an ABL representation is more direct in the last case; we think that this is due to the great deal of overlap among the ideas which led the engineers to such advances in systems design and those which yielded the theory of files.

However, the indicator logic is new only as a method. Most control statements and logical operations written by programmers using COBOL or symbolic machine languages should be considered as a clumsy, approximative and only partially satisfactory replacement for a clean, universal and fully automatic indicator logic.

Let us conclude by pointing out that the advantages of adopting sequential languages does not seem to be bounded to the use of large scale data processing systems. On the contrary, such languages appear to be intimately related to the nature of non-numerical data processing phenomena, regardless of their implementation; for example, a sequential language quite similar to ABL proved to be well suited for representing procedures to be carried out by very simple, externally programmed data processors.

* The preparation of this paper was sponsored by the Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the United States Government. The author also wishes to express his gratitude to M. Alan Feldstein for his help in the preparation of this paper.

References
3. L. Lombardi, "Inexpensive Punched Card Equipment" (Forthcoming).
5. IBM Staff, "Flow Charting and Block Diagramming Techniques" (C20-8008).
6. Notice that Fortran is also a "v. Neumann language".
7. See §, section 4.
8. Another 'sequential language' was conceived independently of us by C.B. Tompkins with the collaboration of M.A. Melkanoff and J.D. Swift.